Tree: data structure made of nodes

- each node has at most one parent
- each node has some number of children

Binary Tree: a tree where each node has at most
  - one right child
  - one left child

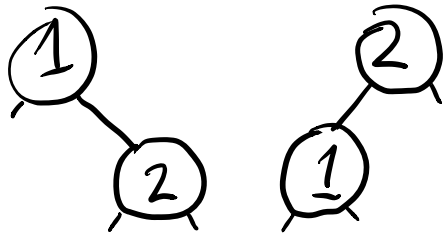
Binary Search Tree: a binary tree where nodes are organized by Key

- all left descendants are smaller
- right larger

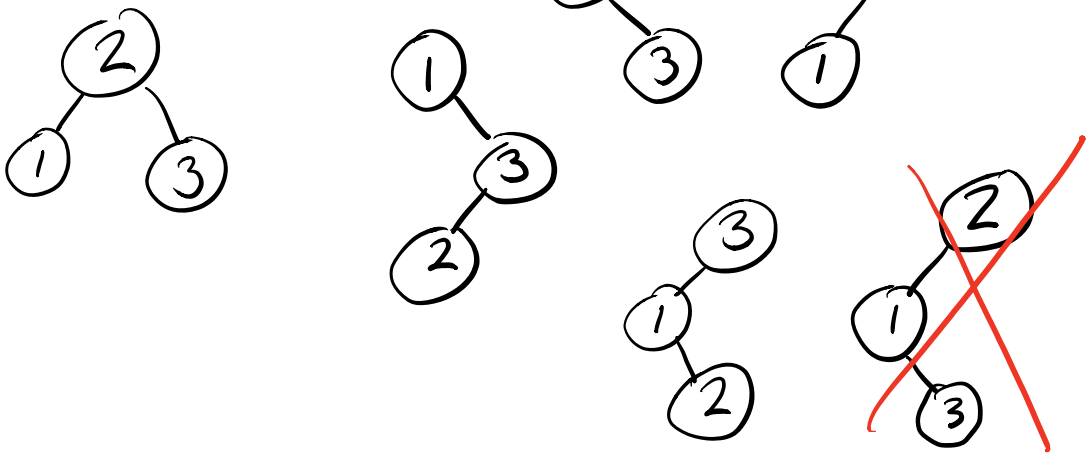
How many possible BSTs are there for keys:

[1]

[1, 2]



[1, 2, 3]

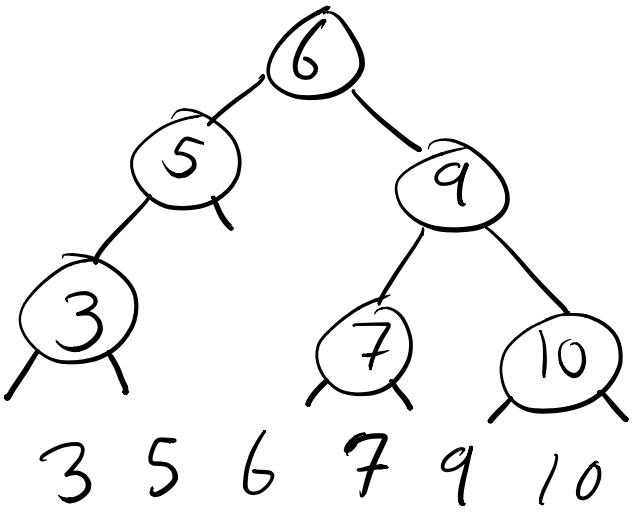


[1, 2, 3, 4] lots

```
method get(key)
method findInSubtree(key, node)

method insert(key, value)
    root = insertInSubtree(key, value, root)
end method

method insertInSubtree(k, v, node)
    if node is null:
        return new BSTNode(k, v)
    if k == node.key:
        ERROR
    if k < node.key:
        node.left = insertInSubtree(k, v, node.left)
    if k > node.key:
        node.right = insertInSubtree(k, v, node.right)
    return node
end method
```



pre: 2, 1, 3

post: 1, 3, 2

3 5 6 7 9 10

```
method InOrderTraversal(node)
    if node is null
        return
    1 InOrderTraversal(node.left)
    2 print(node.key)
    3 InOrderTraversal(node.right)
    return
end method
```

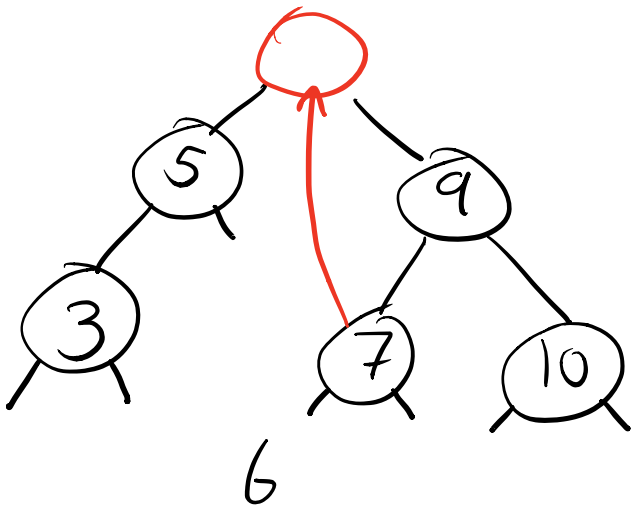
② print
① Left
③ Right

pre-order

① Left
③ Right
② print

post-order

6 5 3 9 7 10



3 5 7 10 9 6

```
method remove(key)
    removeInSubtree(key, root)

end method

method removeInSubtree(key, node)
    if node is null:
        ERROR
    if key < node.key:
node.left = removeInSubtree(key, node.left)
    if key > node.key:
node.right = removeInSubtree(key, node.right)
```

```
if key == node.key:
    if node.right == null
        & node.left == null:
        return null
    else if node.right is null:
        return node.left
    else if node.left is null:
        return node.right
    else:
```